대 한 민 국 특 허 청
KOREAN INTELLECTUAL PROPERTY OFFICE

별첨 사본은 아래 출원의 원본과 동일함을 증명함.

This is to certify that the following application annexed hereto is a true copy from the records of the Korean Intellectual Property Office.

출 원 번 호 : 10-2003-0016206
Application Number

출 원 년 월 일 : 2003년 03월 14일
Date of Application MAR 14, 2003

출 원 인 : 주식회사 안철수연구소 외 1명
Applicant(s) Ahnlab, Inc., et al.

2003 년 04 월 29 일

특 허 청 장
COMMISSIONER

【서지사항】

【서류명】	특허출원서
【권리구분】	특허
【수신처】	특허청장
【참조번호】	0003
【제출일자】	2003.03.14
【국제특허분류】	H04L
【발명의 명칭】	악성 암호화 스크립트에 대한 분석 및 해독 방법
【발명의 영문명칭】	METHOD TO DECRYPT AND ANALYZE THE ENCRYPTED MALICIOUS SCRIPTS
【출원인】	
【명칭】	주식회사 안철수연구소
【출원인코드】	3-1999-902882-2
【출원인】	
【성명】	홍만표
【출원인코드】	4-1999-008549-0
【대리인】	
【성명】	박창남
【대리인코드】	9-2001-000437-2
【포괄위임등록번호】	2003-015642-9
【포괄위임등록번호】	2003-016061-0
【대리인】	
【성명】	진천웅
【대리인코드】	9-1998-000533-6
【포괄위임등록번호】	2003-015641-1
【포괄위임등록번호】	2003-016062-7
【발명자】	
【성명의 국문표기】	이성욱
【성명의 영문표기】	LEE,Sung Wook
【주민등록번호】	690408-1018827
【우편번호】	440-320
【주소】	경기도 수원시 장안구 율전동 샘내마을 삼호아파트 211-906
【국적】	KR

【발명자】

【성명】 홍만표

【출원인코드】 4-1999-008549-0

【발명자】

【성명의 국문표기】 조시행

【성명의 영문표기】 CHO,Si Haeng

【주민등록번호】 620219-1029514

【우편번호】 140-040

【주소】 서울특별시 용산구 산천동 192 리버힐삼성아파트 109-110

【국적】 KR

【공개형태】 간행물 발표

【공개일자】 2002.10.15

【심사청구】 청구

【취지】 특허법 제42조의 규정에 의한 출원, 특허법 제60조의 규정에 의한 출원심사 를 청구합니다. 대리인
박창남 (인) 대리인
진천웅 (인)

【수수료】

【기본출원료】	20	면	29,000 원
【가산출원료】	5	면	5,000 원
【우선권주장료】	0	건	0 원
【심사청구료】	3	항	205,000 원

【합계】 239,000 원

【감면사유】 중소기업

【감면후 수수료】 119,500 원

【첨부서류】 1. 요약서·명세서(도면)_1통 2.중소기업기본법시행령 제2조에의한 중소기업에 해당함을 증명하는 서류_1통 3.공지예외적용대상(신규성상실의예외, 출원시의특례)규정을 적용받기 위한 증명서류_1통

【요약서】

【요약】

본 발명은 악성 암호화 스크립트에 대한 분석 및 해독 방법에 관한 것이다.

이러한 본 발명은 악성 스크립트의 암호화 기법을 해독함수가 존재하면서 해독함수가 외부코드와의 의존성이 없는 독립함수인 경우, 해독함수가 존재하면서 해독함수가 외부코드와의 의존성이 있는 비독립함수인 경우, 해독함수가 존재하지 않는 경우로 구분하여 분류하되, 상기 해독함수가 존재하면서 해독함수가 외부코드와의 의존성이 없는 독립함수인 경우는, 상기 독립함수에 대한 호출식과 함수정의를 추출하는 단계; 추출된 상기 독립함수에 대한 호출식과 함수정의를 실행 또는 에뮬레이션하는 단계; 및 원래의 호출식이 위치한 원본 스크립트에 실행이나 에뮬레이션에 따른 결과값을 대치하여 해독된 스크립트를 취득하는 단계를 포함한 것을 특징으로 한다.

본 발명에 따르면, 해독에 필요한 추가적인 자료없이도 단지 1개의 해독 알고리즘을 통해서 빠르고 용이하게 알려지지 아니한 악성코드도 해독할 수 있다. 또한, 해당 스크립트에서 상수화 할 수 있는 모든 값을 상수로 대치함으로써 암호화된 코드의 해독 뿐만 아니라 추후 코드 분석의 복잡도를 감소시킬 수 있다.

【대표도】

도 4

【색인어】

컴퓨터 바이러스, 암호화, 악성코드, 스크립트, 분석, 해독

【명세서】

【발명의 명칭】

악성 암호화 스크립트에 대한 분석 및 해독 방법{METHOD TO DECRYPT AND ANALYZE THE ENCRYPTED MALICIOUS SCRIPTS}

【도면의 간단한 설명】

도 1 은 본 발명에 따른 악성 스크립트의 암호화 기법의 분류도,

도 2 는 하나의 문자열로 암호화된 악성 스크립트의 실예,

도 3 은 일부 문자열들이 암호화된 악성 스크립트의 실예,

도 4 는 본 발명에 따라 별도의 해독함수가 존재하면서 해독함수가 외부코드와의 의존성이 없는 독립함수인 경우의 암호 해독 절차를 설명하는 흐름도,

도 5 는 도 3 에 제시된 악성 스크립트에서 얻어진 임시 스크립트의 예,

도 6 은 도 5 와 같은 임시 스크립트를 실행한 결과를 나타낸 예,

도 7 은 도 6 과 같은 실행한 결과값을 대치하여 해독된 악성 스크립트를 나타낸 예이다.

【발명의 상세한 설명】

【발명의 목적】

【발명이 속하는 기술분야 및 그 분야의 종래기술】

<8> 본 발명은 악성 암호화 스크립트에 대한 분석 및 해독 방법에 관한 것으로서, 특히 스크립트 암호화 기법에 대한 분석적인 접근을 통해 새로운 암호화 기법의 출현에 대해서도 유연하게 대처할 수 있는 기술에 관한 것이다.

<9> 일반적으로, 암호화(Encryption)는 그 의미가 드러나지 않도록 메시지를 인코딩(Encoding)하는 절차 또는 기법을 의미한다. 그러나, 컴퓨터 바이러스 또는 악성코드에 있어서의 암호화는 악성코드를 변조(Scrambling)함에 의해 바이러스 탐색기(Scanner)가 해당 악성코드의 시그니처(Signature)를 찾지 못하도록 숨기는 기법을 지칭한다. 시그니처란 특정의 악성코드에만 존재하며 다른 프로그램에는 존재하지 않는 짧은 문자열로서, 다른 일반적인 프로그램(Legitimate program)들과 악성코드를 구분하고 그것이 어떠한 악성코드인지를 식별하는데 이용된다. 시그니처를 이용한 악성코드 탐지 방식은 다른 기법에 비해 상대적으로 빠른 속도를 보여주기 때문에 현존하는 대부분의 안티-바이러스(Anti-virus) 제품들이 이러한 시그니처 기반의 감지 방식을 채택하고 있으며, 이에 약간의 휴리스틱(Heuristic) 알고리즘을 결합한 형태가 주종을 이루고 있다.

<10> 그런데, 이러한 감지 방식에 대응하기 위하여, 악성코드 제작자들은 바이러스에 별도의 암호화 기능을 추가하고 있다. 일반적으로, 암호화된 악성코드는 해독루틴과 키값, 그리고 암호화된 악성코드로 구성되고, 실행시에 해독루틴이 먼저 수행된다. 따라서, 해

독루틴은 암호화된 악성코드를 해독하고 해독된 악성코드 쪽으로 제어를 넘김으로써 악성코드가 실행되도록 한다. 이로 인해, 악성코드가 다른 시스템 또는 다른 화일에 자기 복제(Self-replication)를 시도할 때 새로운 키값을 사용하여 인코딩 하는 것만으로 전혀 다른 형태를 보이므로 단순한 스캐닝만으로는 감지할 수 없게 된다.

<11> 한편, 이러한 암호화 바이러스에 대한 대응 기법으로는 엑스-레잉(X-raying)과 에뮬레이션(Emulation) 기법이 사용되고 있다. 엑스-레잉 기법은 해당 악성코드에서 찾아야 하는 시그니처와 악성코드가 사용하는 해독 알고리즘에 대한 사전 지식을 이용하여 그 범위를 좁힌 후에 모든 경우를 시도(Brute-force decryption)하는 방법을 말한다. 즉, 해당 악성코드의 암호화 기법과 시그니처에 대한 모든 정보가 있으며 단지 정확한 키값만을 모르는 경우에 가능한 모든 키값을 이용하여 시그니처가 나타날 수 있는 위치의 문자열을 해독하고 이것이 시그니처와 동일한 값을 가지는지를 검사하여 악성코드 여부를 판단한다. 그러나, 이러한 방법은 찾고자 하는 악성코드의 암호화 기법과 특성이 면밀히 분석되어 충분한 사전 지식이 얻어진 후에 실행이 가능하므로, 알려지지 않은 새로운 악성코드에는 적용하기 어렵다는 단점을 가진다.

<12> 에뮬레이션 기법은 악성코드를 에뮬레이션하여 해독된 코드를 얻는 방법을 말한다. 이진 형태의 악성코드는 해독루틴이 가장 먼저 실행되며, 그 크기가 매우 작기 때문에 가상 기계에서 해당 코드의 일부를 실행함으로써 해독된 악성코드를 얻을 수 있다. 이때, 모든 해독이 완료된 코드를 얻어내려고 한다면 가상기계의 각 메모리 유닛을 감시하여 코드 부분의 메모리에 더 이상의 변화가 일어나지 않는 시점까지 실행을 계속해야 하며, 시그니처 기반의 감지 방법과 결합하여 사용하는 경우에는 시그니처가 위치한 메모리 값들의 해독이 완료되면 그 즉시 에뮬레이션을

중단하고 시그니처 비교를 실시하게 된다. 이같은 특정 시점까지의 완전한(무조건적인) 에뮬레이션 기법은 이진 파일 형태의 악성코드를 해독하는데 효과적이나, 스크립트를 위한 에뮬레이터 구축은 이진 실행 화일에 비해 어려운 것이 현실이다. 즉, 완전한 에뮬레이션을 위해서는 대상 코드가 실행될 수 있는 모든 환경을 가상적으로 만들어 주어야 하는데, 마이크로소프트 비주얼 베이직 스크립트와 같은 스크립트 언어의 경우에는 해당 프로그램에서 사용하는 각종 객체와 제반 환경들을 모두 에뮬레이션 하는 것이 현실적으로 어려우며 부하가 크기 때문이다. 또한, 아무런 위해(Harm) 행위를 하지 않는 일반적인 코드와는 달리, 악성코드는 단순하게 실행시켜 보면서 실행 내역을 프로파일링(Profiling)하는 기법을 사용할 수도 없다.

<13> 결론적으로, 상술한 방법들은 해당 악성코드의 특성과 행위가 알려져 있는 경우에만 적용 가능하거나 이진 화일의 형태를 가진 악성코드의 해독에 적합한 형태이므로, 알려지지 않은 암호화 스크립트에는 적용하기 어려운 것이 사실이다. 따라서, 종래의 스크립트 악성코드에서 사용하는 암호화 기법의 패턴과 해독방법을 정의하고 이를 이용하는 휴리스틱 기반의 방법론이 스크립트 악성코드를 위한 가장 현실적인 암호 해독 기법으로 간주되고 있다. 예컨대, 종래의 많은 비주얼 베이직 스크립트 악성코드는 실제의 악성코드를 하나의 문자열에 암호화하여 놓고 이를 스크립트 언어에서 정의하는 'execute' 문장을 통해 실행시키는 형태로 구성되어 있다. 이러한 경우, 주어진 스크립트에서 'execute' 문장을 찾아 이 문장에서 호출되는 함수를 해독함수로 간주하여 이를 실행

_또는 에뮬레이션함으로써 해독된 악성코드를 얻을 수 있다. 이러한 형태의 암호 해독함수는 상술한 각종 객체와 제반 환경들을 모두 사용하지 않는 기본적인 언어 구조만으로 이루어져 있으며, 프로그램 선두에서 오직 한번만 실행된다. 따라서, 상술한 바와 같은 완전한 에뮬레이션이 필요하지 않게 되며 기본 기능만을 가진 경량의 에뮬레이터만으로도 이러한 형태의 암호 해독 함수를 실행할 수 있으므로 에뮬레이터 구축과 에뮬레이션에 대한 부담은 크지 않게 된다.

<14> 그러나, 휴리스틱 기반의 접근은 새로운 암호화 패턴이 출현할 때마다 이를 처리할 수 있는 코드를 바이러스 탐색기에 추가하여야 하므로, 알려지지 않은 악성 스크립트에 원활하게 대응할 수 없다는 근본적인 문제점을 가지고 있다. 특히, 스크립트 악성코드에서 독특하게 나타나고 있는 문자열 단위의 부분 암호화에 원활하게 대응하기 어렵게 된다.

【발명이 이루고자 하는 기술적 과제】

<15> 이에 본 발명은 상기와 같은 문제점을 해결하기 위하여 안출된 것으로서, 현재의 암호화 기법 뿐만 아니라 향후 등장할 수 있는 기법들까지 포괄 할 수 있는 정확한 분류체제를 수립하고 스크립트 암호화 기법에 대한 분석적인 접근을 통해 새로운 암호화 기법의 출현에 대해서도 유연하게 대처할 수 있는 악성 암호화 스크립트에 대한 분석 및 해독 방법을 제공하는데 그 목적이 있다.

<16> 상기와 같은 목적을 달성하기 위하여 본 발명에 따른 악성 암호화 스크립트에 대한 분석 및 해독 방법은, 악성 스크립트의 암호화 기법을 해독함수가 존재하면서 해독함수가 외부코드와의 의존성이 없는 독립함수인 경우, 해독함수가 존재하면서 해독함수가 외부코드와의 의존성이 있는 비독립함수인 경우, 해독함수가 존재하지 않는 경우로 구분하여 분류하되, 상기 해독함수가 존재하면서 해독함수가 외부코드와의 의존성이 없는 독립함수인 경우는, 상기 독립함수에 대한 호출식과 함수정의를 추출하는 단계; 추출된 상기 독립함수에 대한 호출식과 함수정의를 실행 또는 에뮬레이션하는 단계; 및 원래의 호출식이 위치한 원본 스크립트에 실행이나 에뮬레이션에 따른 결과값을 대치하여 해독된 스크립트를 취득하는 단계를 포함한 것을 특징으로 한다.

<17> 이때, 상기 해독함수의 외부코드와의 의존성 여부는, 해독함수 내부의 모든 코드의 외부코드와의 의존성 여부, 모든 프로그램 내에서 해독함수 호출에 대한 실 인자의 상수 여부, 및 해독함수 내부에서의 부작용이 없는 함수만의 호출 여부에 따라 판단할 수 있는데, 상기 해독함수 내부의 모든 코드의 외부코드와의 의존성 여부는,

<18> n = 스크립트에 정의된 함수의 갯수,

<19> F_i = 스크립트에 i번째로 정의된 함수($1 \le i \le n$),

<20> A_i = 함수 F_i에서 정의 또는 사용된 모든 변수의 집합($1 \le i \le n$),

<21> D_i = 함수 F_i에서 Dim으로 선언된 모든 변수의 집합($1 \le i \le n$), 및

<22> V_o = 어떤 함수에도 속하지 않는 글로벌(global) 영역에서 정의 또는 사용된 변수의 집합이라 할 때,

<23> 함수 F_i 에서 정의 또는 사용된 전역 변수의 집합 V_i 는 다음의 수학식

<24>
$$V_i = A_i - D_i$$
에 따라 구하고,

<25> 함수 F_i의 외부 영역에서 정의 또는 사용된 변수들의 집합 E_i는 다음의 수학식

<26>
$$E_i = \bigcup_{i \neq j, 0 \leq j \leq n} Vj$$
에 따라 구하되,

<27> 함수 F_i 가 다음의 수학식

<28>
$$V_i \cap E_i = \emptyset$$
를 만족할 때,

<29> 함수 F_i 내부의 모든 코드는 외부코드와 의존성이 없다고 판단하여 바람직하게 실시할 수 있다.

【발명의 구성 및 작용】

<30> 이하, 첨부된 도면을 참조하여 본 발명을 상세히 설명하기로 한다.

<31> 도 1 은 본 발명에 따른 악성 스크립트의 암호화 기법의 분류도로서, 암호 해독루틴이 악성코드의 몸체(Body)와 얼마나 깊게 연관되어 있는가에 따른 것이다. 즉, 해독함수의 존재 여부, 및 해독함수가 존재할 경우 해독함수의 외부코드와의 의존성 여부에 따라 분류한다. 따라서, 별도의 해독함수가 존재하면서 해독함수가 외부코드와의 의존성이 없는 독립함수인 경우(이하, '유형 1'이라 함), 별도의 해독함수가 존재하면서 해독함수가 외부코드와의 의존성이 있는 비독립함수인 경우(이하, '유형 2'라 함), 해독함수가

존재하지 않는 경우(이하, '유형 3'이라 함)로 분류할 수 있다. 유형 1은 해독루틴이 가장 독립적으로 존재함으로써 독립적인 실행이 가능한 경우이고, 유형 3은 전체 코드 중 임의의 부분에 삽입되어 프로그램의 실행 상태와 밀접하게 연관된 경우이다.

<32> 한편, 이같은 분류를 좀 더 명확하게 하기 위해서는 단일루틴 내에서 정의되는 코드간의 의존성과는 달리, 해독함수와 외부코드간의 의존성을 정의할 필요가 있다. 즉, 프로그램 코드 내에 정의된 함수 f가 아래와 같은 세 가지 조건을 만족할때 '함수 f가 외부와 의존성이 없다' 또는 '함수 f가 독립적이다' 라고 하고, 이때 '함수 f를 독립함수' 라 한다.

<33> ① 함수 내부의 모든 코드가 외부 코드와 의존성이 없어야 한다. 즉, 함수 f 내에서 정의 또는 사용된 전역 변수들의 집합을 $V(f)$라 하고, 외부에서 정의 또는 사용된 전역 변수들의 집합을 $E(f)$라 할 때, $V(f) \cap E(f) = \varnothing$이어야 한다.

<34> ② 모든 프로그램 내에서, 함수 f 호출에 대한 실 인자(Actual parameter)는 반드시 상수로 주어져야 한다. 변수가 인자로 주어질 경우 이를 통하여 의존성이 발생하기 때문이다.

<35> ③ 함수 내부에서 부작용(Side effect)이 없는 함수만을 호출하여야 한다. 여기에서의 부작용은 I/O 또는 이를 유발하는 모든 행위를 지칭하며, 외부와 의존성이 있는 다른 함수를 실행시켜 간접적으로 의존성을 유발하는 경우까지 포함한다.

<36> 이제, 상기 ① 의 조건에 대해 좀 더 상세히 설명하기로 한다. 특정함수가 독립함수인지를 판단하려면 이것이 사용하는 변수와 함수에 대한 분석이 이루어져야 한다. 독

립함수가 사용할 수 있는 함수는 부작용이 없는 내장 함수와 다른 독립 함수뿐이고, 내장 함수는 언어 사양(Specification)에 명시적으로 정의되어 있으므로 그 중에서 I/O 등의 부작용이 없는 함수들의 리스트를 따로 정의하면 사용유무를 손쉽게 알아낼 수 있다. 그러나, 해당 함수가 지역 변수만을 사용하는 함수인가를 검사하는 작업은 이에 비해 복잡한 과정을 거쳐야 한다.

<37> 모든 변수가 명시적으로 선언된 후 사용되는 일반적인 함수 기반 언어에서는, 각 영역에서 선언된 변수의 집합을 추출하고 해당 함수 내에서 전역 변수를 사용하는지를 검사하는 작업이 비교적 단순하게 이루어진다. 그러나, 비주얼 베이직 스크립트와 같은 인터프리터 기반 스크립트 언어는 일반적인 함수 기반 언어와는 달리 변수의 선언을 반드시 필요로 하지 않는다. 또한, 함수 내부에서 정의되었다 하더라도 Dim 문장을 통해 그것이 지역 변수임을 명시하지 않은 것은 모두 전역 변수로 간주된다. 따라서, 전역 변수의 완전한 리스트를 얻기 위해서는 각각의 함수 내부까지 모든 코드를 검사하여야 한다. 해독 알고리즘을 위한 독립 함수의 존재를 판단할 수 있는 조건은 다음과 같이 정의된다. 먼저, 사용되는 기호를 아래와 같이 정의하기로 한다.

<38> n = 스크립트에 정의된 함수의 갯수

<39> F_i = 스크립트에 i번째로 정의된 함수($1 \le i \le n$)

<40> A_i = 함수 F_i에서 정의 또는 사용된 모든 변수의 집합($1 \le i \le n$)

<41> D_i = 함수 F_i에서 Dim으로 선언된 모든 변수의 집합($1 \le i \le n$)

<42> V_o = 어떤 함수에도 속하지 않는 글로벌(global) 영역에서 정의 또는 사용된 변수의 집합

<43> 이때, 함수 F_i 에서 정의 또는 사용된 전역 변수의 집합 V_i와, 함수 F_i의 외부 영역에서 정의 또는 사용된 변수들의 집합 E_i는 아래의 수학식 1과 수학식 2에 따라 구할 수 있다.

<44>

【수학식 1】 $$V_i = A_i - D_i$$

<45>

【수학식 2】 $$E_i = \bigcup_{i \neq j, 0 \leq j \leq n} Vj$$

<46> 따라서, 함수 F_i 가 독립함수이려면 아래의 수학식 3의 조건을 만족하여야 한다.

<47>

【수학식 3】 $$V_i \cap E_i = \varnothing$$

<48> 즉, 독립함수 F_i 는 자신을 제외한 외부 영역에서 정의 또는 사용된 어떠한 변수도 정의하거나 사용하지 않는 함수이다.

<49> 다른 한편, 실용적인 측면에서 악성 스크립트의 암호화 기법을 살펴 볼 때, 현재의 보편적인 스크립트 악성코드의 암호화 패턴은 크게 두 가지로 분류할 수 있다. 첫번째 패턴은 악성코드 전체가 하나의 문자열로 암호화 되어 있는 경우로서, 도 2 는 VBS/VBSWG.T로 명명된 악성 스크립트를 나타낸다. 비주얼 베이직 스크립트의 'execute' 문장은 인자로 주어진 문자열을 프로그램 코드로 보고 실행하므로, 먼저 해독함수를 호출하게 된다. 따라서, 전체 코드가 완전히 해독된 후에 실행을 개시하게 된다.

<50> 두번째 패턴은 프로그램에서 사용하는 일부 문자열을 암호화한 경우로서, 도 3 은 VBS/TripleSix 로 명명된 악성 스크립트를 나타낸다. 이러한 유형의 악성 스크립트는 하

나 또는 그 이상의 암호 해독 함수를 가지며, 함수의 인자 또는 대입문의 우변값 (r-value) 들에 사용되는 임의의 문자열이 암호화된 형태를 보여준다. 따라서, 첫번째 패턴과는 달리 실행이 진행되면서 필요한 시점에 필요한 문자열이 해독되는 방식으로 동작하게 된다.

<51> 이때, 첫번째 형태의 패턴은 특정한 코드 패턴을 찾는 단순한 접근으로도 해독 가능하나, 두번째 형태의 패턴은 해당 악성 코드에 대한 사전지식 없이 자동적으로 해독 함수를 찾아내는 일이 난해하므로, 새로운 악성 스크립트에 대응하는데 어려움이 따르게 된다. 그러나, 본 발명에 따른 악성 스크립트의 암호화 기법의 분류에 따르면, 모두 유형 1에 속해 동일한 특성을 가지고 있으므로 외형상의 패턴에 관계없이 동일한 방법으로 해독할 수 있다.

<52> 이제, 알려지지 아니한 암호화 스크립트의 해독에 대해 살펴보기로 한다. 암호화된 악성코드의 해독과정에 있어서, 그것이 에뮬레이션 또는 실제 실행에 의하든 어느 정도의 프로그램 실행을 필요로 한다. 그러나, 상술한 바와 같이 완벽한 에뮬레이터의 이용이나 단순한 프로파일링 기법의 사용에는 많은 문제가 있으며, 이러한 문제를 회피하기 위해서는 해당 스크립트의 암호를 해독하는데 필요한 부분만을 발췌하여 실행하는 방법이 필요하게 된다. 특히, 알려지지 않은 새로운 스크립트를 대상으로 한다면 다음과 같은 두 가지 문제에 직면하게 된다.

<53> 첫째, '대상 스크립트가 암호화되어 있는가' 를 판단하는 문제이다. 이미 안티-바이러스 개발자들에 의해 분석된 악성코드는 암호화 여부와 암호 해독 방법이 알려져 있

다. 그러나, 새롭게 등장한 악성코드의 경우에는 아무런 사전 지식 없이 대상 스크립트의 분석만으로 암호화 여부를 판단하여야 한다.

<54> 둘째, '대상 스크립트의 암호 해독 루틴이 어떤 것인가' 를 탐색하는 문제이다. 암호화된 스크립트에는 해독 함수 외에도 많은 함수들이 정의되어 있을 수 있다. 따라서, 새로운 암호화 기법의 출현에 영향 받지 않도록 특정 코드 패턴에 의존하지 않고 해독 함수의 집합을 찾아내야 한다.

<55> 이러한 문제에 대해 본 발명에서 제안하는 해결책은 대상 스크립트가 암호화되었는지, 또는 암호 해독 루틴이 어떤 것인지 분석하기보다는 해당 스크립트에서 상수화 할 수 있는 모든 값을 상수로 대치함으로써 자연적인 암호 해독을 유도하는 것이다. 자신이 악성 행위를 시도 중임을 사용자가 의식하지 못하도록 해야하는 악성 코드의 본질로 인해, 대부분의 악성코드는 특별한 사용자 입력을 받지 않으며 어느 시스템에서나 공통적으로 얻을 수 있는 자원만을 사용하게 된다. 따라서, 악성코드에 입력되는 데이터 집합은 일반 프로그램에 비해 상당히 고정적인 성향을 띄게 되고, 이로 인해 프로그램 내의 많은 변수들이 실제로 매 실행 시마다 동일한 값을 가지게 된다. 특히, 암호화에 관련된 부분은 어떤 상황에서도 원래 의도한 코드를 복원하여야 하므로 이같은 경향이 더욱 두드러지게 나타난다. 따라서, 이러한 기본 방법론은 유형 1의 암호화에 다음과 같이 적용될 수 있다.

<56> 유형 1은 해독함수가 독립함수로 존재하므로, 단순히 독립함수의 실행 결과 값을 대치하는 것만으로 암호화된 내용의 해독이 가능하다. 도 4 는 본 발명에 따라 별도의

해독함수가 존재하면서 해독함수가 외부코드와의 의존성이 없는 독립함수인 경우의 암호 해독 절차를 설명하는 흐름도이다. 도 4 를 참조하면, 독립함수에 대한 호출식(Call expression)과 함수정의를 추출하여, 이것을 실행하거나 에뮬레이션한다(S410,S420). 이어서, 원래의 호출식이 위치한 원본 스크립트에 각각의 실행이나 에뮬레이션에 따른 결과값을 대치하여 해독된 스크립트를 취득한다(S430,S440). 이때, 실제로 암호 해독과는 관계없는 다른 함수도 독립함수로 판정되어 실행될 수 있으나, 이것은 스크립트에 어떠한 문제도 발생시키지 않는다. 독립함수의 리턴값은 정의에 따라 주어진 인자에만 영향을 받으므로 어떤 상태에서 실행하여도 동일한 인자에 대해서 동일한 값이 산출되기 때문이다.

<57> 이번에는, 본 발명의 이해를 돕기 위해서 유형 1과 같은 암호화에 대응하기 위한 해독 알고리즘의 구체적인 동작 사례에 대해 설명하기로 한다. 상술한 바와 같이, 독립함수 호출식의 결과값을 얻는 방법은 실제 실행과 에뮬레이션의 두 가지가 존재하며, 에뮬레이션의 경우에는 에뮬레이터 내부의 자료 구조를 통해 추출된 함수 호출식만을 실행하는 것이 가능하므로, 이하에서는 실제 실행을 통해 결과 값을 얻는 사례를 제시한다.

<58> 먼저, 독립함수가 모두 밝혀지면 이에 대한 호출식을 모두 추출하여 임시 스크립트를 생성한다. 임시 스크립트는 다음과 같은 역할을 수행하는 문장들로 구성하는 것이 바람직하다.

<59> a. 결과값 출력을 위한 화일 개방 및 종결

<60> b. 독립 함수 호출 후 리턴 값을 화일에 기록

<61> c. 독립 함수 정의

<62> d. 타입 핸들링(type handling) 함수 정의

<63> 도 5 는 도 3 에 제시된 악성 스크립트에서 얻어진 임시 스크립트의 예로서, 함수 V와 H가 독립함수로 판정되어 이들에 대한 호출 결과를 화일에 기록하고 있음을 보여준다. 이때, 함수 호출식의 결과값 뿐만 아니라 해당 호출식에 관련된 정보가 함께 기록되는데, 제시된 예에서 함수 호출식의 앞에 기술된 숫자들이 이에 해당한다. 이러한 정보들은 다음 단계에서의 처리에 이용되며, 그 의미는 다음과 같다.

<64> e. 해당 함수 호출식이 있었던 원본 스크립트의 행, 열, 호출식의 문자열 길이

<65> f. 해당 호출을 포함하고 있는 함수의 ID

<66> g. 실행 결과 값의 문자열 길이

<67> h. 실행 결과값

<68> 이러한 정보들 중에서 실행 결과 값의 문자열 길이와 실행 결과값은 타입 핸들링 함수를 통해 얻어진다. 타입 핸들링 함수의 존재는 비주얼 베이직 스크립트의 특성에 기인한 것으로, 비주얼 베이직 스크립트에는 'Variant' 라는 단 한 가지의 타입만이 존재하며 특정값이 주어지면 아래의 표 1 과 같은 부타입(subtype)이 결정된다. 따라서, 비주얼 베이직 스크립트의 함수는 마치 매크로 함수와 같이 매번 다른 타입의 인자를 받을 수도 있으며, 그에 따라 다른 타입의 결과값을 돌려 줄 수도 있다.

<69>

【표 1】

Subtype	Description	Convertile
vbEmpty	Empty (uninitialized)	×
vbNull	Null (no valid data)	×
vbInteger	Integer	O
vbLong	Long integer	O
vbSingle	Single-precision floating-point number	O
vbDouble	Double-precision floating-point number	O
vbCurrency	Currency	O
vbDate	Data	O
vbString	String	O
vbObject	Automation object	×
vbError	Error	×
vbBoolean	Boolean	O
vbVariant	Variant (used only with arrays of Variants)	×
vbDataObject	A data-access object	×
vbByte	Byte	O
vbArray	Array	×

<70> 이러한 비주얼 베이직 스크립트의 특성으로 인해 리턴값의 타입을 실행 전에 확신할 수 없게 되고, 실행 시간에 이를 위한 별도의 처리가 필요하게 된다. 즉, 임시 스크립트를 생성하고 실행하는 것은 그 결과값을 미리 얻어 원본 스크립트의 함수 호출식에 대치하기 위한 것인데, 부타입에 따라 문자열로의 변환이 불가능하여 스크립트에 직접 써넣을 수 없는 무형의 값도 존재하게 된다. 이와 더불어, 문자열로 표현 가능한 부타입이라도 원본 스크립트에 삽입하기 위해서는, 문자열의 양끝에는 따옴표를, Date 타입의 양끝에는 # 을 붙여주고, 기타 숫자형인 경우에는 그대로 문자열로 변환하는 작업을 해주어야 한다.

<71> 이러한 문제의 해결을 위해, 각각의 함수 호출식의 실행 결과는 도 5 에 RunFunc로 나타난 타입 핸들링 함수를 거쳐 적절한 형태의 문자열로 먼저 변환된다. 이때, 문자열로 변환할 수 없는 부타입을 가지는 결과값이 나타나면, 결과값의 길이를 0 으로 기록함으로써 함수 호출식 대치 과정에서 이를 인지하도록 한다.

<72> 이어서, 상기의 과정을 통해 임시 스크립트가 생성되면, 윈도우즈 스크립팅 호스트(Windows Scripting Host)의 호출을 통해 이를 실행시켜 해당 함수의 실행 결과값을 얻는다. 이때, 도 5 와 같이 생성된 임시 스크립트에서 얻어진 실행 결과는 도 6 과 같다. 이렇게 함수 호출식의 결과값이 얻어지면, 이 값들을 원본 스크립트에 대치하여 해독된 스크립트를 얻는다. 상술한 바와 같이 결과값의 문자열 길이가 0 인 것은 이러한 작업에서 제외되며, 또한 모든 함수 호출이 완전하게 대치된 독립함수의 정의 부분은 스크립트 실행에 아무런 영향을 주지 않으므로 삭제될 수 있다. 함수 호출 결과값 대치 이후에 해독된 악성 스크립트는 도 7 과 같다. 즉, 암호화되었던 모든 부분 문자열이 해독되었으며 해독 함수 V와 H의 모든 호출식이 결과값으로 대치되어 함수의 정의가 삭제된 것을 확인할 수 있다.

【발명의 효과】

<73> 이상 설명한 바와 같이, 악성 암호화 스크립트에 대한 분석 및 해독 방법은 해독에 필요한 추가적인 자료없이도 단지 1개의 해독 알고리즘을 통해서 빠르고 용이하게 알려지지 아니한 악성코드도 해독할 수 있다. 또한, 해당 스크립트에서 상수화 할 수 있는 모든 값을 상수로 대치함으로써 암호화된 코드의 해독 뿐만 아니라 추후 코드 분석의 복잡도를 감소시킬 수 있다.

【특허청구범위】

【청구항 1】

암호화된 악성 스크립트를 분석하여 해독하는 방법에 있어서,

악성 스크립트의 암호화 기법을 해독함수가 존재하면서 해독함수가 외부코드와의 의존성이 없는 독립함수인 경우, 해독함수가 존재하면서 해독함수가 외부코드와의 의존성이 있는 비독립함수인 경우, 해독함수가 존재하지 않는 경우로 구분하여 분류하되,

상기 해독함수가 존재하면서 해독함수가 외부코드와의 의존성이 없는 독립함수인 경우는,

상기 독립함수에 대한 호출식과 함수정의를 추출하는 단계;

추출된 상기 독립함수에 대한 호출식과 함수정의를 실행 또는 에뮬레이션하는 단계; 및

원래의 호출식이 위치한 원본 스크립트에 실행이나 에뮬레이션에 따른 결과값을 대치하여 해독된 스크립트를 취득하는 단계를 포함한 것을 특징으로 하는 악성 암호화 스크립트에 대한 분석 및 해독 방법.

【청구항 2】

제 1 항에 있어서 상기 해독함수의 외부코드와의 의존성 여부는,

해독함수 내부의 모든 코드의 외부코드와의 의존성 여부, 모든 프로그램 내에서 해독함수 호출에 대한 실 인자의 상수 여부, 및 해독함수 내부에서의 부작용이 없는 함수만의 호출 여부에 따라 판단하는 것을 특징으로 하는 악성 암호화 스크립트에 대한 분석 및 해독 방법.

【청구항 3】

제 2 항에 있어서 상기 해독함수 내부의 모든 코드의 외부코드와의 의존성 여부는,

n = 스크립트에 정의된 함수의 갯수,

F_i = 스크립트에 i번째로 정의된 함수($1 \leq i \leq n$),

A_i = 함수 F_i에서 정의 또는 사용된 모든 변수의 집합($1 \leq i \leq n$),

D_i = 함수 F_i에서 Dim으로 선언된 모든 변수의 집합($1 \leq i \leq n$), 및

V_o = 어떤 함수에도 속하지 않는 글로벌(global) 영역에서 정의 또는 사용된 변수의 집합이라 할 때,

함수 F_i 에서 정의 또는 사용된 전역 변수의 집합 V_i 는 다음의 수학식

$V_i = A_i - D_i$ 에 따라 구하고,

함수 F_i의 외부 영역에서 정의 또는 사용된 변수들의 집합 E_i는 다음의 수학식

$E_i = \bigcup_{i \neq j, 0 \leq j \leq n} Vj$ 에 따라 구하되,

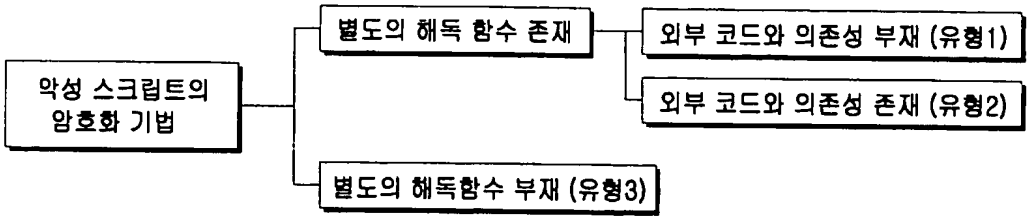
함수 F_i 가 다음의 수학식

$V_i \cap E_i = \emptyset$ 를 만족할 때,

함수 F_i 내부의 모든 코드는 외부코드와 의존성이 없다고 판단하는 것을 특징으로 하는 악성 암호화 스크립트에 대한 분석 및 해독 방법.

【도면】

【도 1】



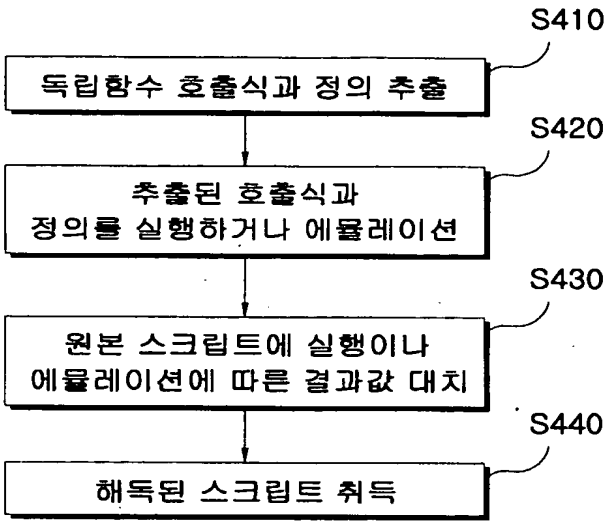
【도 2】

```
Execute(snphhuatvsbkwuj("&Wcr/Wcrvf/H,Vnsl/@w ..... doe!gtobuhno"))

Function snphhuatvsbkwuj(zwbyjntbpmhqqgh)
    For vvpzxfszgnczrao = 1 To Len(zwbyjntbpmhqqgh)
        ccuhbhjhyzkheeq = Mid(zwbyjntbpmhqqgh, vvpzxfszgnczrao, 1)
        If Asc(ccuhbhjhyzkheeq) = 7 Then
            ccuhbhjhyzkheeq =Chr(34)
        End If
        If Asc(ccuhbhjhyzkheeq) <> 35 and Asc(ccuhbhjhyzkheeq) <> 34 Then
            If Asc(ccuhbhjhyzkheeq) Mod 2 = 0 Then
                ccuhbhjhyzkheeq = Chr(Asc(ccuhbhjhyzkheeq) + 1)
            Else
                ccuhbhjhyzkheeq = Chr(Asc(ccuhbhjhyzkheeq) - 1)
            End If
        End If
        snphhuatvsbkwuj = snphhuatvsbkwuj & ccuhbhjhyzkheeq
    Next
End Function
```

【도 3】

```
...
Set Roy = Maggie.CreateTextFile(Maggie.BuildPath(Maggie.GetSpecialFolder(2),V("7594E44554D405E2458545")),True)
Roy.WriteLine(V("E402") & Maggie.BuildPath(Maggie.GetSpecialFolder(2),V("7594E44554D405E245D405")))
Roy.WriteLine("E 0100" & H("4D5AE701630001000600240 6FFFF5F0C"))
Roy.WriteLine("E 0110" & H("000200000001F0FF570000000132504B"))
...
...
Function V(Van)
    For Kirk = 1 To Len(Van) Step 2
        V = V & Chr("&h" & Mid(Van,Kirk + 1,1) & Mid(Van,Kirk,1))
    Next
End Function

Function H(Houten)
    For Luann = 1 To Len(Houten) Step 2
        H = H & " " & Mid(Houten,Luann,2)
    Next
End Function
```

【도 4】



【도 5】

출력 화일 개방
```
Set FSO = CreateObject("Scripting.FileSystemObject")
Set Decrypt = FSO.CreateTextFile("result.txt",True)
```

독립 함수 호출
```
...
Decrypt.WriteLine 23 & " " & 90 & " " & 31 & " " & 3264 & " " & RunFunc (v ( "7594E44554D40513E2241445" ) )
Decrypt.WriteLine 24 & " " & 21 & " " & 25 & " " & 3264 & " " & RunFunc (v ( "04563686F602F66666" ) )
...
```

출력 화일 종결
```
Decrypt.close
```

독립 함수 정의
```
Function V(Van)
...
End Function
Function H(Houten)
...
End Function
```

타입 핸들링 함수 정의
```
Function RunFunc(para)
retValue = para
retType = VarType(retValue)
If (retType >= 2 and retType <=6) or retType = 11 or retType = 17 Then
   RunFunc = Len(CStr(retValue))
ElseIf retType = 7 Then
   retValue = "#" & retValue & "#"
   RunFunc = Len(CStr(retValue))
ElseIf retType = 8 Then
   retValue = Chr(34) & retValue & Chr(34)
   RunFunc = Len(CStr(retValue))
Else
   RunFunc = "0"
End If
If RunFunc <> "0" Then
   RunFunc = Runfunc & " " & retValue
End If
End Function
```

【도 6】

```
  original_expr          return_value
row  col  len funcID   len      string
 22   28   59  3264    28 "Scripting.FileSystemObject"
 23   90   31  3264    14 "WINTEMP1.BAT"
 24   21   25  3264    11 "@echo off"
 25   21   29  3264    13 "debug.exe <"
 25  105   29  3264    13 "WINTEMP.TXT"
 25  140   17  3264     7 " >nul"
...
```

【도 7】

```
...
set maggie = createobject ("Scripting.FileSystemObject")
set marjorie = maggie.createtextfile(maggie.buildpath(maggie.getspecialfolder(2), "WINTEMP1.BAT")), true)
marjorie.writeline("@echo off")
marjorie.writeline("debug.exe <" & maggie.buildpath(maggie.getspecialfolder(2), "WINTEMP.TXT") & " >nul")
marjorie.close
set roy = maggie.createtextfile(maggie.buildpath(maggie.getspecialfolder(2), "WINTEMP.TXT"), true)
roy.writeline("N " & maggie.buildpath(maggie.getspecialfolder(2), "WINTEMP.TMP"))
...
```